ADA - Análisis y Diseño de Algoritmos, 2023-2
Tarea 1: Semanas 1, 2 y 3

Para entregar el domingo 13 de agosto de 2023

Problemas conceptuales a las 23:59 por BrightSpace

Problemas prácticos a las 23:59 en la arena de programación

Tanto los ejercicios como los problemas deben ser resueltos, pero únicamente las soluciones de los problemas deben ser entregadas. La intención de los ejercicios es entrenarlo para que domine el material del curso; a pesar de que no debe entregar soluciones a los ejercicios, usted es responsable del material cubierto en ellos.

Instrucciones para la entrega

Para esta tarea y todas las tareas futuras, la entrega de soluciones es *individual*. Por favor escriba claramente su nombre, código de estudiante y sección en cada hoja impresa entregada o en cada archivo de código (a modo de comentario). Adicionalmente, agregue la información de fecha y nombres de compañeros con los que colaboró; igualmente cite cualquier fuente de información que utilizó.

¿Cómo describir un algoritmo?

En algunos ejercicios y problemas se pide "dar un algoritmo" para resolver un problema. Una solución debe tomar la forma de un pequeño ensayo (es decir, un par de párrafos). En particular, una solución debe resumir en un párrafo el problema y cuáles son los resultados de la solución. Además, se deben incluir párrafos con la siguiente información:

- una descripción del algoritmo en castellano y, si es útil, pseudo-código;
- por lo menos un diagrama o ejemplo que muestre cómo funciona el algoritmo;
- una demostración de la corrección del algoritmo; y
- un análisis de la complejidad temporal del algoritmo.

Recuerde que su objetivo es comunicar claramente un algoritmo. Las soluciones algorítmicas correctas y descritas *claramente* recibirán alta calificación; soluciones complejas, obtusas o mal presentadas recibirán baja calificación.

Ejercicios

La siguiente colección de ejercicios, tomados del libro de Cormen et al. es para repasar y afianzar conceptos, pero no deben ser entregados como parte de la tarea.

1.2-2, 1.2-3 (página 15), 2.2-3 (página 33).

Problemas conceptuales

1. Escribir el código de honor del curso.
2. Problema 1-1: *Comparison of running times* (Cormen et al., página 15).

Problemas prácticos

Hay cinco problemas prácticos cuyos enunciados aparecen a partir de la siguiente página.

A - 10-20-30

Source file name: `10-20-30.py`
Time limit: 1 second

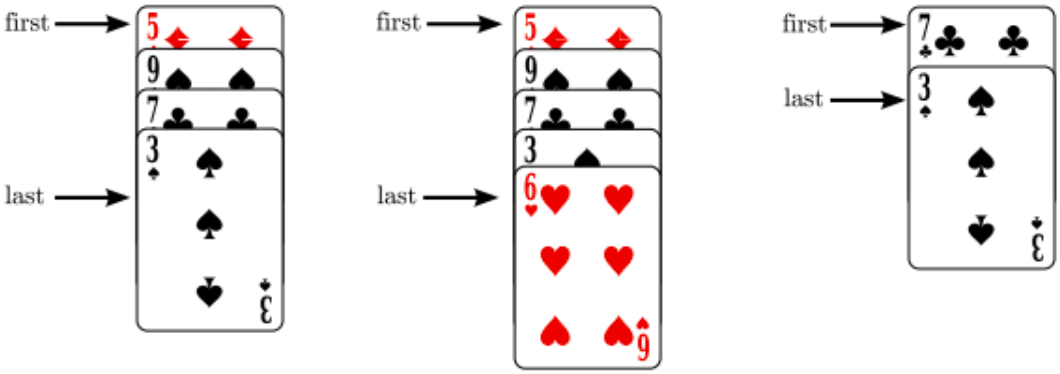
A simple solitaire card game called 10-20-30 uses a standard deck of 52 playing cards in which suit is irrelevant. The value of a face card (king, queen, jack) is 10. The value of an ace is one. The value of each of the other cards is the face value of the card (2, 3, 4, etc.). Cards are dealt from the top of the deck. You begin by dealing out seven cards, left to right forming seven piles. After playing a card on the rightmost pile, the next pile upon which you play a card is the leftmost pile.

For each card placed on a pile, check that pile to see if one of the following three card combinations totals 10, 20, or 30:

1. the first two and last one,
2. the first one and the last two, or
3. the last three cards.

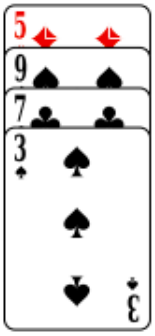
If so, pick up the three cards and place them on the bottom of the deck. For this problem, always check the pile in the order just described. Collect the cards in the order they appear on the pile and put them at the bottom of the deck. Picking up three cards may expose three more cards that can be picked up. If so, pick them up. Continue until no more sets of three can be picked up from the pile.

For example, suppose a pile contains 5 9 7 3 where the 5 is at the first card of the pile, and then a 6 is played. The first two cards plus the last card (5 + 9 + 6) sum to 20. The new contents of the pile after picking up those three cards becomes 7 3. Also, the bottommost card in the deck is now the 6, the card above it is the 9, and the one above the 9 is the 5.



original pile after playing 6 after picking up

If a queen were played instead of the six, 5 + 9 + 10 = 24, and 5 + 3 + 10 = 18, but 7 + 3 + 10 = 20, so the last three cards would be picked up, leaving the pile as 5 9.



original pile

after playing queen

after picking up

If a pile contains only three cards when the three sum to 10, 20, or 30, then the pile "disappears" when the cards are picked up. That is, subsequent play skips over the position that the now-empty pile occupied. You win if all the piles disappear. You lose if you are unable to deal a card. It is also possible to have a draw if neither of the previous two conditions ever occurs.

Write a program that will play games of 10-20-30 given initial card decks as input.

Input

Each input set consists of a sequence of 52 integers separated by spaces and/or ends of line. The integers represent card values of the initial deck for that game. The first integer is the top card of the deck. Input is terminated by a single zero (0) following the last deck.

The input must be read from standard input.

Output

For each input set, print whether the result of the game is a win, loss, or a draw, and print the number of times a card is dealt before the game results can be determined. (A draw occurs as soon as the state of the game is repeated.) Use the format shown in the "Sample Output" section.

The output must be written to standard output.

Sample Input	**Sample Output**
2 6 5 10 10 4 10 10 10 4 5 10 4 5 10 9 7 6 1 7 6 9 5 3 10 10 4 10 9 2 1 10 1 10 10 10 3 10 9 8 10 8 7 1 2 8 6 7 3 3 8 2 4 3 2 10 8 10 6 8 9 5 8 10 5 3 5 4 6 9 9 1 7 6 3 5 10 10 8 10 9 10 10 7 2 6 10 10 4 10 1 3 10 1 1 10 2 2 10 4 10 7 7 10 10 5 4 3 5 7 10 8 2 3 9 10 8 4 5 1 7 6 7 2 6 9 10 2 3 10 3 4 4 9 10 1 1 10 5 10 10 1 8 10 7 8 10 6 10 10 10 9 6 2 10 10 0	Win : 66 Loss: 82 Draw: 73

B - Duathlon

Source file name: `duathlon.py`
Time limit: 1 second

A duathlon is a race that involves running r km and cycling k km. n contestants have entered the race; each contestant has different running and cycling speeds. One of the contestants has bribed the organizers to set r and k so that he can win by the maximum margin. You are to determine if this is possible and, if so, give r and k.

Input

The input contains several sets of input. The description of each set is given below:

The first line of each set contains an integer t, the total distance of the race, in km. That is, $r + k = t$. The next line contains an integer n, the number of competitors. For each contestant, a line follows with two real numbers giving the running and cycling speed for that contestant. The last line of input gives the running and cycling speed of the contestant who has bribed the organizers. You may assume t does not exceed 100 km and n does not exceed 20. Input is terminated by end of file. Two consecutive sets may or may not be separated by a blank line.

The input must be read from standard input.

Output

For each set of input produce one line of output. The output description for each set is given below:

If it is possible to fix the race as describe above, print a message giving r and k accurate to two decimal places, and the amount of seconds by which the cheater will win the race, (0 in case some competitor ties him), as in the sample below. If it is not possible, print `'The cheater cannot win.'`

There is no blank line between outputs for two consecutive sets.

The output must be written to standard output.

Sample Input	**Sample Output**

`100`
`3`
`10.0 40.0`
`20.0 30.0`
`15.0 35.0`
 	
`100`
`3`
`10.0 40.0`
`20.0 30.0`
`15.0 25.0`
	`The cheater can win by 612 seconds with r = 14.29km and k = 85.71km.`
	`The cheater cannot win.`

C - How many inversions?

Source file name: `inversions.py`
Time limit: 1 second

HumbertovMoralov in his student days, he is attended system engineering at "University of missing hill" . He was evaluated in its first course of Analysis of Algorithms (at the first half of 1997) with the following topics and questions:

Inversions:

Let $A[1 \ldots n]$ an array of distinct integers of size n. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an **inversion**of A.

Given the above definition about an inversion, *HmbertovMoralov*must answer the following questions:

1. List all inversions in $\langle 3, 2, 8, 1, 6 \rangle$.
2. What array of size n, with all the numbers from the set 1, 2, 3, . . . , n has the largest amount of inversions? How many inversions?
3. Write an algorithm to determine the number of inversions in any permutation of n elements with $\Theta(n\ logn)$ in the worst case run time.

HumbertovMoralov answered questions 1. and 2. without any problem, but he was not able to solve the question 3 at time. Days later he thought the following solution:

```
1: inv ← 0
2: function MERGE(A[], p, q, r)
3:     n1 ← q−p + 1
4:     n2 ← r−q
5:     create arrays L[1 . . . n1 + 1] and R[1 . . . n2 + 1]
6:     for i = 1 to n1 do
7:         L[i] ← A[p + i−1]
8:     end for
9:     for j = 1 to n2 do
10:        R[j] ← A[q + j]
11:    end for
12:    L[n1 + 1] ← ∞
13:    R[n2 + 1] ← ∞
14:    i ← 1
15:    j ← 1
16:    for k = p to r do
17:        if L[i] ≤ R[j] then
18:            A[k] ← L[i]
19:            i ← i + 1
20:        else
21:            A[k] ← R[j]
22:            j ← j + 1
23:            inv ← inv + n1−i + 1
24:        end if
25:    end for
26: end function
```

```
27: function MergeSort(A[], p, r)
28:     if p < r then
29:         q ← ⌊(p + r)/2⌋
30:         MergeSort(A[], p, q)
31:         MergeSort(A[], q + 1, r)
32:         Merge(A[], p, q, r)
33:     end if
34: end function
```

Will this code solve the problem? Just adding the lines 1 and 23 will be enough to solve the problem?

Please help Humbertov Moralov to validate this solution! For this, you must implement this solution in any of the programming languages accepted by the ACM-ICPC and verify if the expected results are generated.

Input

The input may contain several test cases. Each input case begins with a positive integer n ($1 \leq n \leq 10^6$) denoting the length of A, followed by n distinct lines. Each line contains a positive integer from array A. For $i \in [1, n]$, will meet that $0 \leq A[i] \leq 10^8$. The input ends with a test case in which n is zero, and this case must not be processed.

The input must be read from standard input.

Output

For each test case, your program must print a positive integer representing the total number of inversions in the array A. Each valid test case must generate just one output line.

The output must be written to standard output.

Sample Input	**Sample Output**
5 3 2 8 1 6 5 5 4 3 2 1 1 10 0	5 10 0

D - Lap

Source file name: `lap.py`
Time limit: 1 second

In motorsports it is very common that the leader pilot in a race, at a certain moment, overtakes the last pilot. The leader, at this moment, is one lap ahead of the last pilot, who then becomes a straggler. In this task, given the time it takes for the fastest pilot, and for the slowest pilot, to complete one lap, you have to determine in which lap the slowest pilot will become a straggler. You should consider that, at the beginning, they are side by side at the start line of the circuit, both at the start of lap number 1 (the first lap of the race); and that a new lap always begins right after the leader crosses the start line.

Input

The input contains several test cases. Each test case consists of one line with two integers X and Y ($1 \leq X < Y \leq 2^{50}$), the times, in seconds, that it takes for the fastest and the slowest pilot, respectively, to complete one lap.

The input must be read from standard input.

Output

For each test case in the input program should output line, containing one integer: the lap in which the slowest pilot will become a straggler.

The output must be written to standard output.

Sample Input	**Sample Output**
1 10 4 8 5 7 6875 7109	2 2 4 31

E - Stock Prices

Source file name: `stock.py`
Time limit: 1 second

In this problem we deal with the calculation of stock prices. You need to know the following things about stock prices:

- The *ask price* is the lowest price at which someone is willing to sell a share of a stock.
- The *bid price* is the highest price at which someone is willing to buy a share of a stock.
- The `stock price` is the price at which the last deal was established.

Whenever the bid price is greater than or equal to the ask price, a deal is established. A buy order offering the bid price is matched with a sell order demanding the ask price, and shares are exchanged at the rate of the ask price until either the sell order or the buy order (or both) is fulfilled (i.e., the buyer wants no more stocks, or the seller wants to sell no more stocks). You will be given a list of orders (either buy or sell) and you have to calculate, after each order, the current ask price, bid price and stock price.

Input

On the first line a positive integer: the number of test cases, at most 100. After that per test case:

- One line with an integer n ($1 \leq n \leq 1000$): the number of orders.
- n lines of the form '*order_type* x [`shares at`] y', where *order_type* is either `'buy'` or `'sell'`, x ($1 \leq x \leq 1000$) is the number of shares of a stock someone wishes to buy or to sell, and y ($1 \leq y \leq 1000$) is the desired price.

The input must be read from standard input.

Output

Per test case:

- n lines, each of the form 'a_i b_i s_i', where a_i, b_i and s_i are the current ask, bid and stock prices, respectively, after the i-th order has been processed and all possible deals have taken place. Whenever a price is not defined, output `'-'` instead of the price.

The output must be written to standard output.

Sample Input	**Sample Output**
2 6 buy 10 shares at 100 sell 1 shares at 120 sell 20 shares at 110 buy 30 shares at 110 sell 10 shares at 99 buy 1 shares at 120 6 sell 10 shares at 100 buy 1 shares at 80 buy 20 shares at 90 sell 30 shares at 90 buy 10 shares at 101 sell 1 shares at 80	- 100 - 120 100 - 110 100 - 120 110 110 120 100 99 - 100 120 100 - - 100 80 - 100 90 - 90 80 90 100 80 90 100 - 80